Bayesian Reinforcement Learning

Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart

Abstract This chapter surveys recent lines of work that use Bayesian techniques for reinforcement learning. In Bayesian learning, uncertainty is expressed by a prior distribution over unknown parameters and learning is achieved by computing a posterior distribution based on the data observed. Hence, Bayesian reinforcement learning distinguishes itself from other forms of reinforcement learning by explicitly maintaining a distribution over various quantities such as the parameters of the model, the value function, the policy or its gradient. This yields several benefits: a) domain knowledge can be naturally encoded in the prior distribution to speed up learning; b) the exploration/exploitation tradeoff can be naturally optimized; and c) notions of risk can be naturally taken into account to obtain robust policies.

1 Introduction

Bayesian reinforcement learning is perhaps the oldest form of reinforcement learning. Already in the 1950's and 1960's, several researchers in Operations Research studied the problem of controlling Markov chains with uncertain probabilities. Bellman developed dynamic programing techniques for Bayesian bandit prob-

Nikos Vlassis
Luxembourg Centre for Systems Biomedicine and OneTree Technologies Luxembourg, e-mail: nikos.vlassis@gmail.com

Mohammad Ghavamzadeh
INRIA, e-mail: mohammad.ghavamzadeh@inria.fr

Shie Mannor
Technion, e-mail: shie@ee.technion.ac.il

Pascal Poupart
University of Waterloo, e-mail: ppoupart@cs.uwaterloo.ca

lems (**???**). This work was then generalized to multi-state sequential decision problems with unknown transition probabilities and rewards (**???**). The book "Bayesian Decision Problems and Markov Chains" by **?** gives a good overview of the work of that era. At the time, reinforcement learning was known as *adaptive control processes* and then *Bayesian adaptive control*.

Since Bayesian learning meshes well with decision theory, Bayesian techniques are natural candidates to simultaneously learn about the environment while making decisions. The idea is to treat the unknown parameters as random variables and to maintain an explicit distribution over these variables to quantify the uncertainty. As evidence is gathered, this distribution is updated and decisions can be made simply by integrating out the unknown parameters.

In contrast to traditional reinforcement learning techniques that typically learn point estimates of the parameters, the use of an explicit distribution permits a quantification of the uncertainty that can speed up learning and reduce risk. In particular, the prior distribution allows the practitioner to encode domain knowledge that can reduce the uncertainty. For most real-world problems, reinforcement learning from scratch is intractable since too many parameters would have to be learned if the transition, observation and reward functions are completely unknown. Hence, by encoding domain knowledge in the prior distribution, the amount of interaction with the environment to find a good policy can be reduced significantly. Furthermore, domain knowledge can help avoid catastrophic events that would have to be learned by repeated trials otherwise. An explicit distribution over the parameters also provides a quantification of the uncertainty that is very useful to optimize the exploration/exploitation tradeoff. The choice of action is typically done to maximize future rewards based on the current estimate of the model (exploitation), however there is also a need to explore the uncertain parts of the model in order to refine it and earn higher rewards in the future. Hence, the quantification of this uncertainty by an explicit distribution becomes very useful. Similarly, an explicit quantification of the uncertainty of the future returns can be used to minimize variance or the risk of low rewards.

The chapter is organized as follows. Section 2 describes Bayesian techniques for *model-free* reinforcement learning where explicit distributions over the parameters of the value function, the policy or its gradient are maintained. Section 3 describes Bayesian techniques for *model-based* reinforcement learning, where the distributions are over the parameters of the transition, observation and reward functions. Finally, Section 4 describes Bayesian techniques that take into account the availability of finitely many samples to obtain sample complexity bounds and for optimization under uncertainty.

2 Model-Free Bayesian Reinforcement Learning

Model-free RL methods are those that do not explicitly learn a model of the system and only use sample trajectories obtained by direct interaction with the system.

Model-free techniques are often simpler to implement since they do not require any data structure to represent a model nor any algorithm to update this model. However, it is often more complicated to reason about model-free approaches since it is not always obvious how sample trajectories should be used to update an estimate of the optimal policy or value function. In this section, we describe several Bayesian techniques that treat the value function or policy gradient as random objects drawn from a distribution. More specifically, Section 2.1 describes approaches to learn distributions over Q-functions, Section 2.2 considers distributions over policy gradients and Section 2.3 shows how distributions over value functions can be used to infer distributions over policy gradients in actor-critic algorithms.

2.1 Value-Function Based Algorithms

Value-function based RL methods search in the space of value functions to find the optimal value (action-value) function, and then use it to extract an optimal policy. In this section, we study two Bayesian value-function based RL algorithms: Bayesian Q-learning (**?**) and Gaussian process temporal difference learning (**???**). The first algorithm caters to domains with discrete state and action spaces while the second algorithm handles continuous state and action spaces.

2.1.1 Bayesian Q-learning

Bayesian Q-learning (BQL) (**?**) is a Bayesian approach to the widely-used Q-learning algorithm (**?**), in which exploration and exploitation are balanced by explicitly maintaining a distribution over Q-values to help select actions. Let $D(s,a)$ be a random variable that denotes the sum of discounted rewards received when action a is taken in state s and an optimal policy is followed thereafter. The expectation of this variable $\mathbb{E}[D(s,a)] = Q(s,a)$ is the classic Q-function. In BQL, we place a prior over $D(s,a)$ for any state $s \in \mathscr{S}$ and any action $a \in \mathscr{A}$, and update its posterior when we observe independent samples of $D(s,a)$. The goal in BQL is to learn $Q(s,a)$ by reducing the uncertainty about $\mathbb{E}[D(s,a)]$. BQL makes the following simplifying assumptions: (1) Each $D(s,a)$ follows a normal distribution with mean $\mu(s,a)$ and precision $\tau(s,a)$.[1] This assumption implies that to model our uncertainty about the distribution of $D(s,a)$, it suffices to model a distribution over $\mu(s,a)$ and $\tau(s,a)$. (2) The prior $P(D(s,a))$ for each (s,a)-pair is assumed to be independent and normal-Gamma distributed. This assumption restricts the form of prior knowledge about the system, but ensures that the posterior $P(D(s,a)|d)$ given a sampled sum of discounted rewards $d = \sum_t \gamma^t r(s_t,a_t)$ is also normal-Gamma distributed. However, since the sum of discounted rewards for different (s,a)-pairs are related by Bellman's equation, the posterior distributions become correlated. (3) To

[1] The precision of a Gaussian random variable is the inverse of its variance.

keep the representation simple, the posterior distributions are forced to be independent by breaking the correlations.

In BQL, instead of storing the Q-values as in standard Q-learning, we store the hyper-parameters of the distributions over each $D(s,a)$. Therefore, BQL, in its original form, can only be applied to MDPs with finite state and action spaces. At each time step, after executing a in s and observing r and s', the distributions over the D's are updated as follows:

$$\begin{aligned} P(D(s,a)|r,s') &= \int_d P(D(s,a)|r+\gamma d)P(D(s',a')=d) \\ &\propto \int_d P(D(s,a))P(r+\gamma d|D(s,a))P(D(s',a')=d) \end{aligned}$$

Since the posterior does not have a closed form due to the integral, it is approximated by finding the closest Normal-Gamma distribution by minimizing KL-divergence.

At run-time, it is very tempting to select the action with the highest expected Q-value (i.e., $a^* = \arg\max_a \mathbb{E}[Q(s,a)]$), however this strategy does not ensure exploration. To address this, **?** proposed to add an exploration bonus to the expected Q-values that estimates the myopic *value of perfect information* (VPI).

$$a^* = \arg\max_a \mathbb{E}[Q(s,a)] + VPI(s,a)$$

If exploration leads to a policy change, then the gain in value should be taken into account. Since the agent does not know in advance the effect of each action, VPI is computed as an expected gain

$$VPI(s,a) = \int_{-\infty}^{\infty} dx\, Gain_{s,a}(x)\; P(Q(s,a)=x) \tag{1}$$

where the gain corresponds to the improvement induced by learning the *exact* Q-value (denoted by $q_{s,a}$) of the action executed.

$$Gain_{s,a}(q_{s,a}) = \begin{cases} q_{s,a} - \mathbb{E}[Q(s,a_1)] & \text{if } a \neq a_1 \text{ and } q_{s,a} > \mathbb{E}[Q(s,a_1)] \\ \mathbb{E}[Q(s,a_2)] - q_{s,a} & \text{if } a = a_1 \text{ and } q_{s,a} < \mathbb{E}[Q(s,a_2)] \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

There are two cases: a is revealed to have a higher Q-value than the action a_1 with the highest expected Q-value or the action a_1 with the highest expected Q-value is revealed to have a lower Q-value than the action a_2 with the second highest expected Q-value.

2.1.2 Gaussian Process Temporal Difference Learning

Bayesian Q-learning (BQL) maintains a separate distribution over $D(s,a)$ for each (s,a)-pair, thus, it cannot be used for problems with continuous state or action spaces. **??** proposed a natural extension that uses Gaussian processes. As in BQL,

$D(s,a)$ is assumed to be Normal with mean $\mu(s,a)$ and precision $\tau(s,a)$. However, instead of maintaining a Normal-Gamma over μ and τ simultaneously, a Gaussian over μ is modeled. Since $\mu(s,a) = Q(s,a)$ and the main quantity that we want to learn is the Q-function, it would be fine to maintain a belief only about the mean. To accommodate infinite state and action spaces, a *Gaussian process* is used to model infinitely many Gaussians over $Q(s,a)$ for each (s,a)-pair.

A Gaussian process (e.g., **?**) is the extension of the multivariate Gaussian distribution to infinitely many dimensions or equivalently, corresponds to infinitely many correlated univariate Gaussians. Gaussian processes $GP(\mu,k)$ are parameterized by a mean function $\mu(x)$ and a kernel function $k(x,x')$ which are the limit of the mean vector and covariance matrix of multivariate Gaussians when the number of dimensions become infinite. Gaussian processes are often used for functional regression based on sampled realizations of some unknown underlying function.

Along those lines, **??** proposed a *Gaussian Process Temporal Difference* (GPTD) approach to learn the Q-function of a policy based on samples of discounted sums of returns. Recall that the distribution of the sum of discounted rewards for a fixed policy π is defined recursively as follows:

$$D(\mathbf{z}) = r(\mathbf{z}) + \gamma D(\mathbf{z}') \qquad \text{where} \quad \mathbf{z}' \sim P^{\pi}(\mathbf{z}'|\mathbf{z}). \tag{3}$$

When $\mathbf{z}$ refers to states then $\mathbb{E}[D] = V$ and when it refers to state-action pairs then $\mathbb{E}[D] = Q$. Unless otherwise specified, we will assume that $\mathbf{z} = (s,a)$. We can decompose D as the sum of its mean Q and a zero-mean noise term ΔQ, which will allow us to place a distribution directly over Q later on. Replacing $D(\mathbf{z})$ by $Q(\mathbf{z}) + \Delta Q(\mathbf{z})$ in Eq. 3 and grouping the ΔQ terms into a single zero-mean noise term $N(\mathbf{z},\mathbf{z}') = \Delta Q(\mathbf{z}) - \gamma \Delta Q(\mathbf{z}')$, we obtain

$$r(\mathbf{z}) = Q(\mathbf{z}) - \gamma Q(\mathbf{z}') + N(\mathbf{z},\mathbf{z}') \qquad \text{where} \quad \mathbf{z}' \sim P^{\pi}(\mathbf{z}'|\mathbf{z}). \tag{4}$$

The GPTD learning model (**??**) is based on the statistical generative model in Eq. 4 that relates the observed reward signal r to the unobserved action-value function Q. Now suppose that we observe the sequence $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t$, then Eq. 4 leads to a system of t equations that can be expressed in matrix form as

$$r_{t-1} = H_t Q_t + N_t \ , \tag{5}$$

where

$$\begin{aligned} r_t &= \big(r(\mathbf{z}_0), \ldots, r(\mathbf{z}_t)\big)^{\top}, & Q_t &= \big(Q(\mathbf{z}_0), \ldots, Q(\mathbf{z}_t)\big)^{\top}, \\ N_t &= \big(N(\mathbf{z}_0,\mathbf{z}_1), \ldots, N(\mathbf{z}_{t-1},\mathbf{z}_t)\big)^{\top}, \end{aligned} \tag{6}$$

$$H_t = \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & \ldots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \ldots & 1 & -\gamma \end{bmatrix}. \tag{7}$$

If we assume that the residuals $\Delta Q(\mathbf{z}_0),\ldots,\Delta Q(\mathbf{z}_t)$ are zero-mean Gaussians with variance σ^2, and moreover, each residual is generated independently of all the others, i.e., $\mathbb{E}[\Delta Q(\mathbf{z}_i)\Delta Q(\mathbf{z}_j)]=0$, for $i\neq j$, it is easy to show that the noise vector N_t is Gaussian with mean 0 and the covariance matrix

$$\Sigma_t=\sigma^2 H_t H_t^\top=\sigma^2\begin{bmatrix}1+\gamma^2 & -\gamma & 0 & \ldots & 0\\ -\gamma & 1+\gamma^2 & -\gamma & \ldots & 0\\ \vdots & \vdots & & & \vdots\\ 0 & 0 & \ldots & -\gamma & 1+\gamma^2\end{bmatrix}. \tag{8}$$

In episodic tasks, if $\mathbf{z}_{t-1}$ is the last state-action pair in the episode (i.e., s_t is a zero-reward absorbing terminal state), H_t becomes a square $t\times t$ invertible matrix of the form shown in Eq. 7 with its last column removed. The effect on the noise covariance matrix Σ_t is that the bottom-right element becomes 1 instead of $1+\gamma^2$.

Placing a GP prior $GP(0,k)$ on Q, we may use Bayes' rule to obtain the moments $\hat{Q}$ and $\hat{k}$ of the posterior Gaussian process on Q:

$$\begin{aligned}\hat{Q}_t(\mathbf{z})&=\mathbb{E}\left[Q(\mathbf{z})|\mathscr{D}_t\right]=k_t(\mathbf{z})^\top\alpha_t,\\ \hat{k}_t(\mathbf{z},\mathbf{z}')&=\mathbf{Cov}\left[Q(\mathbf{z}),Q(\mathbf{z}')|\mathscr{D}_t\right]=k(\mathbf{z},\mathbf{z}')-k_t(\mathbf{z})^\top C_t k_t(\mathbf{z}'),\end{aligned} \tag{9}$$

where $\mathscr{D}_t$ denotes the observed data up to and including time step t. We used here the following definitions:

$$\begin{aligned}k_t(\mathbf{z})&=\big(k(\mathbf{z}_0,\mathbf{z}),\ldots,k(\mathbf{z}_t,\mathbf{z})\big)^\top, & K_t&=\big[k_t(\mathbf{z}_0),k_t(\mathbf{z}_1),\ldots,k_t(\mathbf{z}_t)\big],\\ \alpha_t&=H_t^\top\left(H_tK_tH_t^\top+\Sigma_t\right)^{-1}r_{t-1}\,, & C_t&=H_t^\top\left(H_tK_tH_t^\top+\Sigma_t\right)^{-1}H_t\,.\end{aligned} \tag{10}$$

As more samples are observed, the posterior covariance decreases, reflecting a growing confidence in the Q-function estimate $\hat{Q}_t$.

The GPTD model described above is kernel-based and non-parametric. It is also possible to employ a parametric representation under very similar assumptions. In the parametric setting, the GP Q is assumed to consist of a linear combination of a finite number of basis functions: $Q(\cdot,\cdot)=\phi(\cdot,\cdot)^\top W$, where ϕ is the feature vector and W is the weight vector. In the parametric GPTD, the randomness in Q is due to W being a random vector. In this model, we place a Gaussian prior over W and apply Bayes' rule to calculate the posterior distribution of W conditioned on the observed data. The posterior mean and covariance of Q may be easily computed by multiplying the posterior moments of W with the feature vector ϕ. See **?** for more details on parametric GPTD.

In the parametric case, the computation of the posterior may be performed online in $O(n^2)$ time per sample and $O(n^2)$ memory, where n is the number of basis functions used to approximate Q. In the non-parametric case, we have a new basis function for each new sample we observe, making the cost of adding the t'th sample $O(t^2)$ in both time and memory. This would seem to make the non-parametric form of GPTD computationally infeasible except in small and simple problems. How-

ever, the computational cost of non-parametric GPTD can be reduced by using an online sparsification method (e.g., **?**), to a level that it can be efficiently implemented online.

The choice of the prior distribution may significantly affect the performance of GPTD. However, in the standard GPTD, the prior is set at the beginning and remains unchanged during the execution of the algorithm. **?** developed an online model selection method for GPTD using sequential MC techniques, called *replacing-kernel RL*, and empirically showed that it yields better performance than the standard GPTD for many different kernel families.

Finally, the GPTD model can be used to derive a SARSA-type algorithm, called GPSARSA (**??**), in which state-action values are estimated using GPTD and policies are improved by a ε-greedily strategy while slowly decreasing ε toward 0. The GPTD framework, especially the GPSARSA algorithm, has been successfully applied to large scale RL problems such as the control of an octopus arm (**?**) and wireless network association control (**?**).

2.2 Policy Gradient Algorithms

Policy gradient (PG) methods are RL algorithms that maintain a parameterized action-selection policy and update the policy parameters by moving them in the direction of an estimate of the gradient of a performance measure (e.g., **???**). These algorithms have been theoretically and empirically analyzed (e.g., **??**), and also extended to POMDPs (**?**). However, both the theoretical results and empirical evaluations have highlighted a major shortcoming of these algorithms, namely, the high variance of the gradient estimates.

Several solutions have been proposed for this problem such as: (1) To use an artificial *discount factor* ($0 < \gamma < 1$) in these algorithms (**??**). However, this creates another problem by introducing bias into the gradient estimates. (2) To subtract a *reinforcement baseline* from the average reward estimate in the updates of PG algorithms (**????**). This approach does not involve biasing the gradient estimate, however, what would be a good choice for a state-dependent baseline is more or less an open question. (3) To replace the policy gradient estimate with an estimate of the so-called *natural* policy gradient (**???**). In terms of the policy update rule, the move to a natural-gradient rule amounts to linearly transforming the gradient using the inverse Fisher information matrix of the policy. In empirical evaluations, natural PG has been shown to significantly outperform conventional PG (**????**).

However, both conventional and natural policy gradient methods rely on Monte-Carlo (MC) techniques in estimating the gradient of the performance measure. Although MC estimates are unbiased, they tend to suffer from high variance, or alternatively, require excessive sample sizes (see **?** for a discussion). In the case of policy gradient estimation this is exacerbated by the fact that consistent policy improvement requires multiple gradient estimation steps. **?** proposes a Bayesian alternative to MC estimation of an integral, called *Bayesian quadrature* (BQ). The

idea is to model integrals of the form $\int dx\ f(x)g(x)$ as random quantities. This is done by treating the first term in the integrand, f, as a random function over which we express a prior in the form of a Gaussian process (GP). Observing (possibly noisy) samples of f at a set of points $\{x_1, x_2, \ldots, x_M\}$ allows us to employ Bayes' rule to compute a posterior distribution of f conditioned on these samples. This, in turn, induces a posterior distribution over the value of the integral. **?** experimentally demonstrated how this approach, when applied to the evaluation of an expectation, can outperform MC estimation by orders of magnitude, in terms of the mean-squared error. Interestingly, BQ is often effective even when f is known. The posterior of f can be viewed as an approximation of f (that converges to f in the limit), but this approximation can be used to perform the integration in closed form. In contrast, MC integration uses the exact f, but only at the points sampled. So BQ makes better use of the information provided by the samples by using the posterior to "interpolate" between the samples and by performing the integration in closed form.

In this section, we study a Bayesian framework for policy gradient estimation based on modeling the policy gradient as a GP (**?**). This reduces the number of samples needed to obtain accurate gradient estimates. Moreover, estimates of the natural gradient as well as a measure of the uncertainty in the gradient estimates, namely, the gradient covariance, are provided at little extra cost.

Let us begin with some definitions and notations. A *stationary policy* $\pi(\cdot|s)$ is a probability distribution over actions, conditioned on the current state. Given a fixed policy π, the MDP induces a Markov chain over state-action pairs, whose transition probability from (s_t, a_t) to (s_{t+1}, a_{t+1}) is $\pi(a_{t+1}|s_{t+1})P(s_{t+1}|s_t, a_t)$. We generically denote by $\xi = (s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T)$, $T \in \{0, 1, \ldots, \infty\}$ a path generated by this Markov chain. The probability (density) of such a path is given by

$$P(\xi|\pi) = P_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t). \tag{11}$$

We denote by $R(\xi) = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$ the discounted *cumulative return* of the path ξ, where $\gamma \in [0, 1]$ is a discount factor. $R(\xi)$ is a random variable both because the path ξ itself is a random variable, and because, even for a given path, each of the rewards sampled in it may be stochastic. The expected value of $R(\xi)$ for a given path ξ is denoted by $\bar{R}(\xi)$. Finally, we define the *expected return* of policy π as

$$\eta(\pi) = \mathbb{E}[R(\xi)] = \int d\xi\ \bar{R}(\xi) P(\xi|\pi). \tag{12}$$

In PG methods, we define a class of smoothly parameterized stochastic policies $\{\pi(\cdot|s;\theta), s \in \mathscr{S}, \theta \in \Theta\}$. We estimate the gradient of the expected return w.r.t. the policy parameters θ, from the observed system trajectories. We then improve the policy by adjusting the parameters in the direction of the gradient. We use the following equation to estimate the gradient of the expected return:

$$\nabla\eta(\theta) = \int d\xi\ \bar{R}(\xi)\frac{\nabla P(\xi;\theta)}{P(\xi;\theta)}P(\xi;\theta), \tag{13}$$

where $\frac{\nabla P(\xi;\theta)}{P(\xi;\theta)} = \nabla \log P(\xi;\theta)$ is called the *score function* or *likelihood ratio*. Since the initial-state distribution P_0 and the state-transition distribution P are independent of the policy parameters θ, we may write the score function of a path ξ using Eq. 11 as[2]

$$u(\xi;\theta) = \frac{\nabla P(\xi;\theta)}{P(\xi;\theta)} = \sum_{t=0}^{T-1}\frac{\nabla\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta)} = \sum_{t=0}^{T-1}\nabla\log\pi(a_t|s_t;\theta). \tag{14}$$

The frequentist approach to PG uses classical MC to estimate the gradient in Eq. 13. This method generates i.i.d. sample paths $\xi_1,\ldots,\xi_M$ according to $P(\xi;\theta)$, and estimates the gradient $\nabla\eta(\theta)$ using the MC estimator

$$\widehat{\nabla\eta}(\theta) = \frac{1}{M}\sum_{i=1}^{M}R(\xi_i)\nabla\log P(\xi_i;\theta) = \frac{1}{M}\sum_{i=1}^{M}R(\xi_i)\sum_{t=0}^{T_i-1}\nabla\log\pi(a_{t,i}|s_{t,i};\theta). \tag{15}$$

This is an unbiased estimate, and therefore, by the law of large numbers, $\widehat{\nabla\eta}(\theta)\to\nabla\eta(\theta)$ as M goes to infinity, with probability one.

In the frequentist approach to PG, the performance measure used is $\eta(\theta)$. In order to serve as a useful performance measure, it has to be a deterministic function of the policy parameters θ. This is achieved by averaging the cumulative return $R(\xi)$ over all possible paths ξ and all possible returns accumulated in each path. In the Bayesian approach we have an additional source of randomness, namely, our subjective Bayesian uncertainty concerning the process generating the cumulative return. Let us denote $\eta_B(\theta) = \int d\xi\ \bar{R}(\xi)P(\xi;\theta)$, where $\eta_B(\theta)$ is a random variable because of the Bayesian uncertainty. We are interested in evaluating the posterior distribution of the *gradient* of $\eta_B(\theta)$ w.r.t. the policy parameters θ. The posterior mean of the gradient is

$$\mathbb{E}\left[\nabla\eta_B(\theta)|\mathscr{D}_M\right] = \mathbb{E}\left[\int d\xi\ R(\xi)\frac{\nabla P(\xi;\theta)}{P(\xi;\theta)}P(\xi;\theta)\Big|\mathscr{D}_M\right]. \tag{16}$$

In the Bayesian policy gradient (BPG) method of **?**, the problem of estimating the gradient of the expected return (Eq. 16) is cast as an integral evaluation problem, and then the BQ method (**?**), described above, is used. In BQ, we need to partition the integrand into two parts, $f(\xi;\theta)$ and $g(\xi;\theta)$. We will model f as a GP and assume that g is a function known to us. We will then proceed by calculating the posterior moments of the gradient $\nabla\eta_B(\theta)$ conditioned on the observed data $\mathscr{D}_M = \{\xi_1,\ldots,\xi_M\}$. Because in general, $R(\xi)$ cannot be known exactly, even for a given ξ (due to the stochasticity of the rewards), $R(\xi)$ should always belong to the GP part of the model, i.e., $f(\xi;\theta)$. **?** proposed two different ways of partitioning the integrand in Eq. 16, resulting in two distinct Bayesian models. Table 1 in **?** summarizes the two

[2] To simplify notation, we omit ∇ and u's dependence on the policy parameters θ, and use ∇ and $u(\xi)$ in place of ∇_θ and $u(\xi;\theta)$ in the sequel.

models. Models 1 and 2 use Fisher-type kernels for the prior covariance of f. The choice of Fisher-type kernels was motivated by the notion that a good representation should depend on the data generating process (see **??** for a thorough discussion). The particular choices of linear and quadratic Fisher kernels were guided by the requirement that the posterior moments of the gradient be analytically tractable.

Models 1 and 2 can be used to define algorithms for evaluating the gradient of the expected return w.r.t. the policy parameters. The algorithm (for either model) takes a set of policy parameters θ and a sample size M as input, and returns an estimate of the posterior moments of the gradient of the expected return. This Bayesian PG evaluation algorithm, in turn, can be used to derive a Bayesian policy gradient (BPG) algorithm that starts with an initial vector of policy parameters θ_0 and updates the parameters in the direction of the posterior mean of the gradient of the expected return, computed by the Bayesian PG evaluation procedure. This is repeated N times, or alternatively, until the gradient estimate is sufficiently close to zero.

As mentioned earlier, the kernel functions used in Models 1 and 2 are both based on the Fisher information matrix $G(\theta)$. Consequently, every time we update the policy parameters we need to recompute G. In most practical situations, G is not known and needs to be estimated. **?** described two possible approaches to this problem: MC estimation of G and maximum likelihood (ML) estimation of the MDP's dynamics and use it to calculate G. They empirically showed that even when G is estimated using MC or ML, BPG performs better than MC-based PG algorithms.

BPG may be made significantly more efficient, both in time and memory, by sparsifying the solution. Such sparsification may be performed incrementally, and helps to numerically stabilize the algorithm when the kernel matrix is singular, or nearly so. Similar to the GPTD case, one possibility is to use the on-line sparsification method proposed by **?** to selectively add a new observed path to a set of *dictionary* paths, which are used as a basis for approximating the full solution. Finally, it is easy to show that the BPG models and algorithms can be extended to POMDPs along the same lines as in **?**.

2.3 Actor-Critic Algorithms

Actor-critic (AC) methods were among the earliest to be investigated in RL (**??**). They comprise a family of RL methods that maintain two distinct algorithmic components: an *actor*, whose role is to maintain and update an action-selection policy; and a *critic*, whose role is to estimate the value function associated with the actor's policy. A common practice is that the actor updates the policy parameters using stochastic gradient ascent, and the critic estimates the value function using some form of temporal difference (TD) learning (**?**). When the representations used for the actor and the critic are *compatible*, in the sense explained in **?** and **?**, the resulting AC algorithm is simple, elegant, and provably convergent (under appropriate conditions) to a local maximum of the performance measure used by the critic plus a measure of the TD error inherent in the function approximation scheme (**??**). The

apparent advantage of AC algorithms (e.g., **????**) over PG methods, which avoid using a critic, is that using a critic tends to reduce the variance of the policy gradient estimates, making the search in policy-space more efficient and reliable.

Most AC algorithms are based on parametric critics that are updated to optimize frequentist fitness criteria. However, the GPTD model described in Section 2.1, provides us with a Bayesian class of critics that return a full posterior distribution over value functions. In this section, we study a Bayesian actor-critic (BAC) algorithm that incorporates GPTD in its critic (**?**). We show how the posterior moments returned by the GPTD critic allow us to obtain closed-form expressions for the posterior moments of the policy gradient. This is made possible by utilizing the Fisher kernel (**?**) as our prior covariance kernel for the GPTD state-action *advantage* values. This is a natural extension of the BPG approach described in Section 2.2. It is important to note that while in BPG the basic observable unit, upon which learning and inference are based, is a complete trajectory, BAC takes advantage of the Markov property of the system trajectories and uses individual state-action-reward transitions as its basic observable unit. This helps reduce variance in the gradient estimates, resulting in steeper learning curves compared to BPG and the classic MC approach.

Under certain regularity conditions (**?**), the expected return of a policy π defined by Eq. 12 can be written as

$$\eta(\pi) = \int_{\mathscr{Z}} d\mathbf{z}\, \mu^{\pi}(\mathbf{z}) \bar{r}(\mathbf{z}),$$

where $\bar{r}(\mathbf{z})$ is the mean reward for the state-action pair $\mathbf{z}$, and $\mu^{\pi}(\mathbf{z}) = \sum_{t=0}^{\infty} \gamma^t P_t^{\pi}(\mathbf{z})$ is a discounted weighting of state-action pairs encountered while following policy π. Integrating a out of $\mu^{\pi}(\mathbf{z}) = \mu^{\pi}(s,a)$ results in the corresponding discounted weighting of states encountered by following policy π; $\rho^{\pi}(s) = \int_{\mathscr{A}} da \mu^{\pi}(s,a)$. Unlike ρ^{π} and μ^{π}, $(1-\gamma)\rho^{\pi}$ and $(1-\gamma)\mu^{\pi}$ are distributions. They are analogous to the stationary distributions over states and state-action pairs of policy π in the undiscounted setting, since as $\gamma \to 1$, they tend to these stationary distributions, if they exist. The policy gradient theorem (**?**, Proposition 1; **?**, Theorem 1; **?**, Theorem 1) states that the gradient of the expected return for parameterized policies is given by

$$\nabla\eta(\theta) = \int dsda\, \rho(s;\theta) \nabla\pi(a|s;\theta) Q(s,a;\theta) = \int d\mathbf{z}\, \mu(\mathbf{z};\theta) \nabla \log \pi(a|s;\theta) Q(\mathbf{z};\theta). \tag{17}$$

Observe that if $b : \mathscr{S} \to \mathbb{R}$ is an arbitrary function of s (also called a *baseline*), then

$$\begin{aligned} \int_{\mathscr{Z}} dsda\, \rho(s;\theta) \nabla\pi(a|s;\theta) b(s) &= \int_{\mathscr{S}} ds\, \rho(s;\theta) b(s) \nabla\Big(\int_{\mathscr{A}} da\, \pi(a|s;\theta) \Big) \\ &= \int_{\mathscr{S}} ds\, \rho(s;\theta) b(s) \nabla\big(1\big) = 0, \end{aligned}$$

and thus, for any baseline $b(s)$, Eq. 17 may be written as

$$\nabla\eta(\theta) = \int_{\mathscr{Z}} d\mathbf{z}\, \mu(\mathbf{z};\theta)\nabla\log\pi(a|s;\theta)[Q(\mathbf{z};\theta)+b(s)]. \tag{18}$$

Now consider the case in which the action-value function for a fixed policy π, Q^π, is approximated by a learned function approximator. If the approximation is sufficiently good, we may hope to use it in place of Q^π in Eqs. 17 and 18, and still point roughly in the direction of the true gradient. **?** and **?** showed that if the approximation $\hat{Q}^\pi(\cdot;\mathbf{w})$ with parameter $\mathbf{w}$ is *compatible*, i.e., $\nabla_{\mathbf{w}}\hat{Q}^\pi(s,a;\mathbf{w}) = \nabla\log\pi(a|s;\theta)$, and if it minimizes the mean squared error

$$\mathscr{E}^\pi(\mathbf{w}) = \int_{\mathscr{Z}} d\mathbf{z}\, \mu^\pi(\mathbf{z})\big[Q^\pi(\mathbf{z}) - \hat{Q}^\pi(\mathbf{z};\mathbf{w})\big]^2 \tag{19}$$

for parameter value $\mathbf{w}^*$, then we may replace Q^π with $\hat{Q}^\pi(\cdot;\mathbf{w}^*)$ in Eqs. 17 and 18. An approximation for the action-value function, in terms of a linear combination of basis functions, may be written as $\hat{Q}^\pi(\mathbf{z};\mathbf{w}) = \mathbf{w}^\top\psi(\mathbf{z})$. This approximation is compatible if the ψ's are compatible with the policy, i.e., $\psi(\mathbf{z};\theta) = \nabla\log\pi(a|s;\theta)$. It can be shown that the mean squared-error problems of Eq. 19 and

$$\mathscr{E}^\pi(\mathbf{w}) = \int_{\mathscr{Z}} d\mathbf{z}\, \mu^\pi(\mathbf{z})\big[Q^\pi(\mathbf{z}) - \mathbf{w}^\top\psi(\mathbf{z}) - b(s)\big]^2 \tag{20}$$

have the same solutions (e.g., **??**), and if the parameter $\mathbf{w}$ is set to be equal to $\mathbf{w}^*$ in Eq. 20, then the resulting mean squared error $\mathscr{E}^\pi(\mathbf{w}^*)$ is further minimized by setting $b(s) = V^\pi(s)$ (**??**). In other words, the variance in the action-value function estimator is minimized if the baseline is chosen to be the value function itself. This means that it is more meaningful to consider $\mathbf{w}^{*\top}\psi(\mathbf{z})$ as the least-squared optimal parametric representation for the *advantage* function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ rather than the action-value function $Q^\pi(s,a)$.

We are now in a position to describe the main idea behind the BAC approach. Making use of the linearity of Eq. 17 in Q and denoting $g(\mathbf{z};\theta) = \mu^\pi(\mathbf{z})\nabla\log\pi(a|s;\theta)$, we obtain the following expressions for the posterior moments of the policy gradient (**?**):

$$\begin{aligned}
\mathbb{E}[\nabla\eta(\theta)|\mathscr{D}_t] &= \int_{\mathscr{Z}} d\mathbf{z}\, g(\mathbf{z};\theta)\hat{Q}_t(\mathbf{z};\theta) = \int_{\mathscr{Z}} d\mathbf{z}\, g(\mathbf{z};\theta)k_t(\mathbf{z})^\top\alpha_t\ , \\
\mathbf{Cov}\,[\nabla\eta(\theta)|\mathscr{D}_t] &= \int_{\mathscr{Z}^2} d\mathbf{z}\, d\mathbf{z}' g(\mathbf{z};\theta)\hat{S}_t(\mathbf{z},\mathbf{z}')g(\mathbf{z}';\theta)^\top \\
&= \int_{\mathscr{Z}^2} d\mathbf{z}\, d\mathbf{z}' g(\mathbf{z};\theta)\Big(k(\mathbf{z},\mathbf{z}') - k_t(\mathbf{z})^\top C_t k_t(\mathbf{z}')\Big)\, g(\mathbf{z}';\theta)^\top,
\end{aligned} \tag{21}$$

where $\hat{Q}_t$ and $\hat{S}_t$ are the posterior moments of Q computed by the GPTD critic from Eq. 9.

These equations provide us with the general form of the posterior policy gradient moments. We are now left with a computational issue, namely, how to compute the following integrals appearing in these expressions?

$$U_t = \int_{\mathscr{Z}} d\mathbf{z}\, g(\mathbf{z};\theta) k_t(\mathbf{z})^\top \quad \text{and} \quad V = \int_{\mathscr{Z}^2} d\mathbf{z} d\mathbf{z}'\, g(\mathbf{z};\theta) k(\mathbf{z},\mathbf{z}') g(\mathbf{z}';\theta)^\top. \quad (22)$$

Using the definitions in Eq. 22, we may write the gradient posterior moments compactly as

$$\mathbb{E}[\nabla\eta(\theta)|\mathscr{D}_t] = U_t\alpha_t \qquad \text{and} \qquad \mathbf{Cov}[\nabla\eta(\theta)|\mathscr{D}_t] = V - U_t C_t U_t^\top. \quad (23)$$

? showed that in order to render these integrals analytically tractable, the prior covariance kernel should be defined as $k(\mathbf{z},\mathbf{z}') = k_s(s,s') + k_F(\mathbf{z},\mathbf{z}')$, the sum of an arbitrary state-kernel k_s and the Fisher kernel between state-action pairs $k_F(\mathbf{z},\mathbf{z}') = u(\mathbf{z})^\top G(\theta)^{-1} u(\mathbf{z}')$. They proved that using this prior covariance kernel, U_t and V from Eq. 22 satisfy $U_t = [u(\mathbf{z}_0), \ldots, u(\mathbf{z}_t)]$ and $V = G(\theta)$. When the posterior moments of the gradient of the expected return are available, a Bayesian actor-critic (BAC) algorithm can be easily derived by updating the policy parameters in the direction of the mean.

Similar to the BPG case in Section 2.2, the Fisher information matrix of each policy may be estimated using MC or ML methods, and the algorithm may be made significantly more efficient, both in time and memory, and more numerically stable by sparsifying the solution using for example the online sparsification method of **?**.

3 Model-Based Bayesian Reinforcement Learning

In model-based RL we explicitly estimate a model of the environment dynamics while interacting with the system. In model-based Bayesian RL we start with a prior belief over the unknown parameters of the MDP model. Then, when a realization of an unknown parameter is observed while interacting with the environment, we update the belief to reflect the observed data. In the case of discrete state-action MDPs, each unknown transition probability $P(s'|s,a)$ is an unknown parameter $\theta_a^{s,s'}$ that takes values in the $[0,1]$ interval; consequently beliefs are probability densities over continuous intervals. Model-based approaches tend to be more complex computationally than model-free ones, but they allow for prior knowledge of the environment to be more naturally incorporated in the learning process.

3.1 POMDP formulation of Bayesian RL

We can formulate model-based Bayesian RL as a partially observable Markov decision process (POMDP) (**?**), which is formally described by a tuple $\langle \mathscr{S}_P, \mathscr{A}_P, \mathscr{O}_P, T_P, Z_P, R_P \rangle$. Here $\mathscr{S}_P = \mathscr{S} \times \{\theta_a^{s,s'}\}$ is the hybrid set of states defined by the cross product of the (discrete and fully observable) nominal MDP states s and the (continuous and unobserved) model parameters $\theta_a^{s,s'}$ (one parameter for each feasible state-

action-state transition of the MDP). The action space of the POMDP $\mathscr{A}_P = \mathscr{A}$ is the same as that of the MDP. The observation space $\mathscr{O}_P = \mathscr{S}$ coincides with the MDP state space since the latter is fully observable. The transition function $T_P(s,\theta,a,s',\theta') = P(s',\theta'|s,\theta,a)$ can be factored in two conditional distributions, one for the MDP states $P(s'|s,\theta_a^{s,s'},a) = \theta_a^{s,s'}$, and one for the unknown parameters $P(\theta'|\theta) = \delta_\theta(\theta')$ where $\delta_\theta(\theta')$ is a Kronecker delta with value 1 when $\theta' = \theta$ and value 0 otherwise). This Kronecker delta reflects the assumption that unknown parameters are stationary, i.e., θ does not change with time. The observation function $Z_P(s',\theta',a,o) = P(o|s',\theta',a)$ indicates the probability of making an observation o when joint state s',θ' is reached after executing action a. Since the observations are the MDP states, then $P(o|s',\theta',a) = \delta_{s'}(o)$.

We can formulate a *belief-state MDP* over this POMDP by defining beliefs over the unknown parameters $\theta_a^{s,s'}$. The key point is that this belief-state MDP is *fully observable* even though the original RL problem involves hidden quantities. This formulation effectively turns the reinforcement learning problem into a planning problem in the space of beliefs over the unknown MDP model parameters.

For discrete MDPs a natural representation of beliefs is via Dirichlet distributions, as Dirichlets are conjugate densities of multinomials (**?**). A Dirichlet distribution $Dir(p;n) \propto \Pi_i p_i^{n_i-1}$ over a multinomial p is parameterized by positive numbers n_i, such that $n_i - 1$ can be interpreted as the number of times that the p_i-probability event has been observed. Since each feasible transition s,a,s' pertains only to one of the unknowns, we can model beliefs as products of Dirichlets, one for each unknown model parameter $\theta_a^{s,s'}$.

Belief monitoring in this POMDP corresponds to Bayesian updating of the beliefs based on observed state transitions. For a *prior* belief $b(\theta) = Dir(\theta;n)$ over some transition parameter θ, when a specific (s,a,s') transition is observed in the environment, the *posterior* belief is analytically computed by the Bayes' rule, $b'(\theta) \propto \theta_a^{s,s'} b(\theta)$. If we represent belief states by a tuple $\langle s, \{n_a^{s,s'}\}\rangle$ consisting of the current state s and the hyperparameters $n_a^{s,s'}$ for each Dirichlet, belief updating simply amounts to setting the current state to s' and incrementing by one the hyperparameter $n_a^{s,s'}$ that matches the observed transition s,a,s'.

The POMDP formulation of Bayesian reinforcement learning provides a natural framework to reason about the exploration/exploitation tradeoff. Since beliefs encode all the information gained by the learner (i.e., sufficient statistics of the history of past actions and observations) and an optimal POMDP policy is a mapping from beliefs to actions that maximizes the expected total rewards, it follows that an optimal POMDP policy naturally optimizes the exploration/exploitation tradeoff. In other words, since the goal in balancing exploitation (immediate gain) and exploration (information gain) is to maximize the overall sum of rewards, then the best tradeoff is achieved by the best POMDP policy. Note however that this assumes that the prior belief is accurate and that computation is exact, which is rarely the case in practice. Nevertheless, the POMDP formulation provides a useful formalism to design algorithms that naturally tradeoff the exploration/exploitation tradeoff.

The POMDP formulation reduces the RL problem to a planning problem with special structure. In the next section we derive the parameterization of the optimal value function, which can be computed exactly by dynamic programming (**?**). However, since the complexity grows exponentially with the planning horizon, we also discuss some approximations.

3.2 Bayesian RL via Dynamic Programming

Using the fact that POMDP observations in Bayesian RL correspond to nominal MDP states, Bellman's equation for the optimal value function in the belief-state MDP reads (**?**)

$$V_s^*(b) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,b,a)\, V_{s'}^*(b_a^{s,s'}). \tag{24}$$

Here s is the current nominal MDP state, b is the current belief over the model parameters θ, and $b_a^{s,s'}$ is the updated belief after transition s,a,s'. The transition model is defined as

$$P(s'|s,b,a) = \int_\theta d\theta\; b(\theta)\; P(s'|s,\theta,a) = \int_\theta d\theta\; b(\theta)\; \theta_a^{s,s'}, \tag{25}$$

and is just the average transition probability $P(s'|s,a)$ with respect to belief b. Since an optimal POMDP policy achieves by definition the highest attainable expected future reward, it follows that such a policy would automatically optimize the exploration/exploitation tradeoff in the original RL problem.

It is known (see, e.g., chapter **??** in this book) that the optimal finite-horizon value function of a POMDP with discrete states and actions is piecewise linear and convex, and it corresponds to the upper envelope of a set Γ of linear segments called α-vectors: $V^*(b) = \max_{\alpha\in\Gamma} \alpha(b)$. In the literature, α is both defined as a linear function of b (i.e., $\alpha(b)$) and as a vector of s (i.e., $\alpha(s)$) such that $\alpha(b) = \sum_s b(s)\alpha(s)$. Hence, for discrete POMDPs, value functions can be parameterized by a set of α-vectors each represented as a vector of values for each state. Conveniently, this parameterization is closed under Bellman backups.

In the case of Bayesian RL, despite the hybrid nature of the state space, the piecewise linearity and convexity of the value function may still hold as demonstrated by **?** and **?**. In particular, the optimal finite-horizon value function of a discrete-action POMDP corresponds to the upper envelope of a set Γ of linear segments called α-functions (due to the continuous nature of the POMDP state θ), which can be grouped in subsets per nominal state s:

$$V_s^*(b) = \max_{\alpha\in\Gamma} \alpha_s(b). \tag{26}$$

Here α can be defined as a linear function of b subscripted by s (i.e., $\alpha_s(b)$) or as a function of θ subscripted by s (i.e., $\alpha_s(\theta)$) such that

$$\alpha_s(b) = \int_\theta d\theta \; b(\theta) \; \alpha_s(\theta). \tag{27}$$

Hence value functions in Bayesian RL can also be parameterized as a set of α-functions. Moreover, similarly to discrete POMDPs, the α-functions can be updated by Dynamic Programming (DP) as we will show next. However, in Bayesian RL the representation of α-functions grows in complexity with the number of DP backups: For horizon T, the optimal value function may involve a number of α-functions that is exponential in T, but also each α-function will have a representation complexity (for instance, number of nonzero coefficients in a basis function expansion) that is also exponential in T, as we will see next.

3.2.1 Value function parameterization

Suppose that the optimal value function $V_s^k(b)$ for k steps-to-go is composed of a set Γ^k of α-functions such that $V_s^k(b) = \max_{\alpha\in\Gamma^k} \alpha_s(b)$. Using Bellman's equation, we can compute by dynamic programming the best set Γ^{k+1} representing the optimal value function V^{k+1} with $k+1$ stages-to-go. First we rewrite Bellman's equation (Eq. 24) by substituting V^k for the maximum over the α-functions in Γ^k as in Eq. 26:

$$V_s^{k+1}(b) = \max_a R(b,a) + \gamma \sum_{s'} P(s'|s,b,a) \max_{\alpha\in\Gamma^k} \alpha_{s'}(b_a^{s,s'}).$$

Then we decompose Bellman's equation in three steps. The first step finds the maximal α-function for each a and s'. The second step finds the best action a. The third step performs the actual Bellman backup using the maximal action and α-functions:

$$\alpha_{b,a}^{s,s'} = \arg\max_{\alpha\in\Gamma^k} \alpha(b_a^{s,s'}) \tag{28}$$

$$a_b^s = \arg\max_a R(s,a) + \gamma \sum_{s'} P(s'|s,b,a) \alpha_{b,a}^{s,s'}(b_a^{s,s'}) \tag{29}$$

$$V_s^{k+1}(b) = R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,b,a_b^s) \alpha_{b,a_b^s}^{s,s'}(b_{a_b^s}^{s,s'}) \tag{30}$$

We can further rewrite the third step by using α-functions in terms of θ (instead of b) and expanding the belief state $b_{a_b^s}^{s,s'}$:

$$V_s^{k+1}(b) = R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,b,a_b^s) \int_\theta d\theta \; b_{a_b^s}^{s,s'}(\theta) \alpha_{b,a_b^s}^{s,s'}(\theta) \tag{31}$$

$$= R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,b,a_b^s) \int_\theta d\theta \; \frac{b(\theta)P(s'|s,\theta,a_b^s)}{P(s'|s,b,a_b^s)} \alpha_{b,a_b^s}^{s,s'}(\theta) \tag{32}$$

$$= R(s,a_b^s) + \gamma \sum_{s'} \int_\theta d\theta \; b(\theta)P(s'|s,\theta,a_b^s) \alpha_{b,a_b^s}^{s,s'}(\theta) \tag{33}$$

$$= \int_\theta d\theta \; b(\theta)[R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,\theta,a_b^s) \alpha_{b,a_b^s}^{s,s'}(\theta)] \tag{34}$$

The expression in square brackets is a function of s and θ, so we can use it as the definition of an α-function in Γ^{k+1}:

$$\alpha_{b,s}(\theta) = R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,\theta,a_b^s) \alpha_{b,a_b^s}^{s,s'}(\theta). \tag{35}$$

For every b we define such an α-function, and together all $\alpha_{b,s}$ form the set Γ^{k+1}. Since each $\alpha_{b,s}$ was defined by using the optimal action and α-functions in Γ^k, it follows that each $\alpha_{b,s}$ is necessarily optimal at b and we can introduce a max over all α-functions with no loss:

$$V_s^{k+1}(b) = \int_\theta d\theta \; b(\theta)\alpha_{b,s}(\theta) = \alpha_s(b) = \max_{\alpha \in \Gamma^{k+1}} \alpha_s(b). \tag{36}$$

Based on the above we can show the following (we refer to the original paper for the proof):

Theorem 1 (?). *The α-functions in Bayesian RL are linear combinations of products of (unnormalized) Dirichlets.*

Note that in this approach the representation of α-functions grows in complexity with the number of DP backups: Using the above theorem and Eq. 35, one can see that the number of components of each α-function grow in each backup by a factor $O(|\mathscr{S}|)$, which yields a number of components that grows exponentially with the planning horizon. In order to mitigate the exponential growth in the number of components, we can project linear combinations of components onto a smaller number of components (e.g., a monomial basis). **?** describe various projection schemes that achieve that.

3.2.2 Exact and approximate DP algorithms

Having derived a representation for α-functions that is closed under Bellman backups, one can now transfer several of the algorithms for discrete POMDPs to Bayesian RL. For instance, one can compute an optimal finite-horizon Bayesian RL controller by resorting to a POMDP solution technique akin to Monahan's enumeration algorithm (see chapter **??** in this book), however in each backup the number of supporting α-functions will in general be an exponential function of $|\mathscr{S}|$.

Alternatively, one can devise approximate (point-based) value iteration algorithms that exploit the value function parameterization via α-functions. For instance, **?** proposed the BEETLE algorithm for Bayesian RL, which is an extension of the Perseus algorithm for discrete POMDPs (**?**). In this algorithm, a set of reachable (s,b) pairs is sampled by simulating several runs of a random policy. Then (approximate) value iteration is done by performing point-based backups at the sampled (s,b) pairs, pertaining to the particular parameterization of the α-functions.

The use of α-functions in value iteration allows for the design of *offline* (i.e., pre-compiled) solvers, as the α-function parameterization offers a generalization to off-sample regions of the belief space. BEETLE is the only known algorithm in the literature that exploits the form of the α-functions to achieve generalization in model-based Bayesian RL. Alternatively, one can use any generic function approximator. For instance, **?** describes and actor-critic algorithms that approximates the value function with a linear combination of features in (s,θ). Most other model-based Bayesian RL algorithms are *online* solvers that do not explicitly parameterize the value function. We briefly describe some of these algorithms next.

3.3 Approximate Online Algorithms

Online algorithms attempt to approximate the Bayes optimal action by reasoning over the current belief, which often results in myopic action selection strategies. This approach avoids the overhead of offline planning (as with BEETLE), but it may require extensive deliberation at runtime that can be prohibitive in practice.

Early approximate online RL algorithms were based on confidence intervals (**???**) or the value of perfect information (VPI) criterion for action selection (**?**), both resulting in myopic action selection strategies. The latter involves estimating the distribution of optimal Q-values for the MDPs in the support of the current belief, which are then used to compute the expected 'gain' for switching from one action to another, hopefully better, action. Instead of building an explicit distribution over Q-values (as in Section 2.1.1), we can use the distribution over models $P(\theta)$ to sample models and compute the optimal Q-values of each model. This yields a sample of Q-values that approximates the underlying distribution over Q-values. The exploration gain of each action can then be estimated according to Eq. 2, where the expectation over Q-values is approximated by the sample mean. Similar to Eq. 1, the value of perfect information can be approximated by:

$$VPI(s,a) \approx \frac{1}{\sum_i w_\theta^i} \sum_i w_\theta^i Gain_{s,a}(q_{s,a}^i) \tag{37}$$

where the w_θ^i's are the importance weights of the sampled models depending on the proposal distribution used. **?** describe several efficient procedures to sample the models from some proposal distributions that may be easier to work with than $P(\theta)$.

An alternative myopic Bayesian action selection strategy is *Thompson sampling*, which involves sampling just one MDP from the current belief, solve this MDP to optimality (e.g., by Dynamic Programming), and execute the optimal action at the current state (**??**), a strategy that reportedly tends to over-explore (**?**).

One may achieve a less myopic action selection strategy by trying to compute a near-optimal policy in the belief-state MDP of the POMDP (see previous section). Since this is just an MDP (albeit continuous and with a special structure), one may use any approximate solver for MDPs. **??** have pursued this idea by applying the sparse sampling algorithm of **?** on the belief-state MDP. This approach carries out an explicit lookahead to the effective horizon starting from the current belief, backing up rewards through the tree by dynamic programming or linear programming (**?**), resulting in a near-Bayes-optimal exploratory action. The search through the tree does not produce a policy that will generalize over the belief space however, and a new tree will have to be generated at each time step which can be expensive in practice. Presumably the sparse sampling approach can be combined with an approach that generalizes over the belief space via an α-function parameterization as in BEETLE, although no algorithm of that type has been reported so far.

3.4 Bayesian Multi-Task Reinforcement Learning

Multi-task learning (MTL) is an important learning paradigm and has recently been an area of active research in machine learning (e.g., **??**). A common setup is that there are multiple related tasks for which we are interested in improving the performance over individual learning by sharing information across the tasks. This transfer of information is particularly important when we are provided with only a limited number of data to learn each task. Exploiting data from related problems provides more training samples for the learner and can improve the performance of the resulting solution. More formally, the main objective in MTL is to maximize the improvement over individual learning averaged over the tasks. This should be distinguished from transfer learning in which the goal is to learn a suitable bias for a class of tasks in order to maximize the expected future performance.

Most RL algorithms often need a large number of samples to solve a problem and cannot directly take advantage of the information coming from other similar tasks. However, recent work has shown that transfer and multi-task learning techniques can be employed in RL to reduce the number of samples needed to achieve nearly-optimal solutions. All approaches to multi-task RL (MTRL) assume that the tasks share similarity in some components of the problem such as dynamics, reward structure, or value function. While some methods explicitly assume that the shared components are drawn from a common generative model (**???**), this assumption is more implicit in others (**??**). In **?**, tasks share the same dynamics and reward features, and only differ in the weights of the reward function. The proposed method initializes the value function for a new task using the previously learned value functions as a prior. **?** and **?** both assume that the distribution over some components of

the tasks is drawn from a hierarchical Bayesian model (HBM). We describe these two methods in more details below.

? study the MTRL scenario in which the learner is provided with a number of MDPs with common state and action spaces. For any given policy, only a small number of samples can be generated in each MDP, which may not be enough to accurately evaluate the policy. In such a MTRL problem, it is necessary to identify classes of tasks with similar structure and to learn them jointly. It is important to note that here a task is a pair of MDP and policy such that all the MDPs have the same state and action spaces. They consider a particular class of MTRL problems in which the tasks *share structure in their value functions*. To allow the value functions to share a common structure, it is assumed that they are all sampled from a common prior. They adopt the GPTD value function model (see Section 2.1) for each task, model the distribution over the value functions using a HBM, and develop solutions to the following problems: (i) joint learning of the value functions (multi-task learning), and (ii) efficient transfer of the information acquired in (i) to facilitate learning the value function of a newly observed task (transfer learning). They first present a HBM for the case in which all the value functions belong to the same class, and derive an EM algorithm to find MAP estimates of the value functions and the model's hyper-parameters. However, if the functions do not belong to the same class, simply learning them together can be detrimental (*negative transfer*). It is therefore important to have models that will generally benefit from related tasks and will not hurt performance when the tasks are unrelated. This is particularly important in RL as changing the policy at each step of policy iteration (this is true even for fitted value iteration) can change the way tasks are clustered together. This means that even if we start with value functions that all belong to the same class, after one iteration the new value functions may be clustered into several classes. To address this issue, they introduce a Dirichlet process (DP) based HBM for the case that the value functions belong to an undefined number of classes, and derive inference algorithms for both the multi-task and transfer learning scenarios in this model.

The MTRL approach in **?** also uses a DP-based HBM to model the distribution over a common structure of the tasks. In this work, the tasks *share structure in their dynamics and reward function*. The setting is incremental, i.e., the tasks are observed as a sequence, and there is no restriction on the number of samples generated by each task. The focus is not on joint learning with finite number of samples, it is on using the information gained from the previous tasks to facilitate learning in a new one. In other words, the focus in this work is on transfer and not on multi-task learning.

3.5 Incorporating Prior Knowledge

When transfer learning and multi-task learning are not possible, the learner may still want to use domain knowledge to reduce the complexity of the learning task. In non-Bayesian reinforcement learning, domain knowledge is often implicitly encoded in the choice of features used to encode the state space, parametric form of the value

function, or the class of policies considered. In Bayesian reinforcement learning, the prior distribution provides an explicit and expressive mechanism to encode domain knowledge. Instead of starting with a non-informative prior (e.g., uniform, Jeffrey's prior), one can reduce the need for data by specifying a prior that biases the learning towards parameters that a domain expert feels are more likely.

For instance, in model-based Bayesian reinforcement learning, Dirichlet distributions over the transition and reward distributions can naturally encode an expert's bias. Recall that the hyperparameters $n_i - 1$ of a Dirichlet can be interpreted as the number of times that the p_i-probability event has been observed. Hence, if the expert has access to prior data where each event occured $n_i - 1$ times or has reasons to believe that each event would occur $n_i - 1$ times in a fictitious experiment, then a corresponding Dirichlet can be used as an informative prior. Alternatively, if one has some belief or prior data to estimate the mean and variance of some unknown multinomial, then the hyperparameters of the Dirichlet can be set by moment matching.

A drawback of the Dirichlet distribution is that it only allows unimodal priors to be expressed. However, mixtures of Dirichlets can be used to express multimodal distributions. In fact, since Dirichlets are monomials (i.e., $Dir(\theta) = \prod_i \theta_i^{n_i}$), then mixtures of Dirichlets are polynomials with positive coefficients (i.e., $\sum_j c_j \prod_i \theta_i^{n_{ij}}$). So with a lage enough number of mixture components it is possible to approximate arbitrarily closely any desirable prior over an unknown multinomial distribution. **?** explored the use of mixtures of Dirichlets to express joint priors over the model dynamics and the policy. Although mixtures of Dirichlets are quite expressive, in some situation it may be possible to structure the priors according to a generative model. To that effect, **?** explored the use of hierarchical priors such as hierarchical Dirichlet processes over the model dynamics and policies represented as stochastic finite state controllers. The multi-task and transfer learning techniques described in the previous section also explore hierarchical priors over the value function (**?**) and the model dynamics (**?**).

4 Finite Sample Analysis and Complexity Issues

One of the main attractive features of the Bayesian approach to RL is the possibility of obtaining finite sample estimation for the statistics of a given policy in terms of posterior expected value and variance. This idea was first pursued by **?**, who considered the bias and variance of the value function estimate of a single policy. Assuming an exogenous sampling process (i.e., we only get to observe the transitions and rewards, but not to control them), there exists a nominal model (obtained by, say, maximum a-posteriori probability estimate) and a posterior probability distribution over all possible models. Given a policy π and a posterior distribution over model $\theta =< T, r >$, we can consider the expected posterior value function as:

$$\mathbb{E}_{\tilde{T},\tilde{r}}\left[\mathbb{E}_s[\sum_{t=1}^{\infty}\gamma^t\tilde{r}(s_t)|\tilde{T}]\right], \tag{38}$$

where the outer expectation is according to the posterior over the parameters of the MDP model and the inner expectation is with respect to transitions given that the model parameters are fixed. Collecting the infinite sum, we get

$$\mathbb{E}_{\tilde{T},\tilde{r}}\left[(I-\gamma\tilde{T}_\pi)^{-1}\tilde{r}_\pi\right], \tag{39}$$

where $\tilde{T}_\pi$ and $\tilde{r}_\pi$ are the transition matrix and reward vector of policy π when model $<\tilde{T},\tilde{r}>$ is the true model. This problem maximizes the expected return over both the trajectories and the model random variables. Because of the nonlinear effect of $\tilde{T}$ on the expected return, **?** argue that evaluating the objective of this problem for a given policy is already difficult.

Assuming a Dirichlet prior for the transitions and a Gaussian prior for the rewards, one can obtain bias and variance estimates for the value function of a given policy. These estimates are based on first order or second order approximations of Equation (39). From a computational perspective, these estimates can be easily computed and the value function can be de-biased. When trying to optimize over the policy space, **?** show experimentally that the common approach consisting of using the most likely (or expected) parameters leads to a strong bias in the performance estimate of the resulting policy.

The Bayesian view for a finite sample naturally leads to the question of policy optimization, where an additional maximum over all policies is taken in (38). The standard approach in Markov decision processes is to consider the so-called robust approach: assume the parameters of the problem belong to some uncertainty set and find the policy with the best worst-case performance. This can be done efficiently using dynamic programming style algorithms; see **??**. The problem with the robust approach is that it leads to over-conservative solutions. Moreover, the currently available algorithms require the uncertainty in different states to be uncorrelated, meaning that the uncertainty set is effectively taken as the Cartesian product of state-wise uncertainty sets.

One of the benefits of the Bayesian perspective is that it enables using certain risk aware approaches since we have a probability distribution on the available models. For example, it is possible to consider bias-variance tradeoffs in this context, where one would maximize reward subject to variance constraints or give a penalty for excessive variance. Mean-variance optimization in the Bayesian setup seems like a difficult problem, and there are currently no known complexity results about it. Curtailing this problem, **?** present an approximation to a risk-sensitive percentile optimization criterion:

$$\begin{aligned} &\text{maximize}_{y\in\mathbb{R},\pi\in\Gamma} && y \\ &\text{s.t. } P_\theta\left(\mathbb{E}_s(\textstyle\sum_{t=0}^{\infty}\gamma^t r_t(s_t)|s_0\propto q,\pi)\geq y\right)\geq 1-\varepsilon. \end{aligned} \tag{40}$$

For a given policy π, the above chance-constrained problem gives us a $1-\varepsilon$ guarantee that π will perform better than the computed y. The parameter ε in Equation (40) measures the risk of the policy doing worse than y. The performance measure we use is related to risk-sensitive criteria often used in finance such as value-at-risk.

The program (40) is not as conservative as the robust approach (which is derived by taking $\varepsilon = 0$), but also not as optimistic as taking the nominal parameters. From a computational perspective, **?** show that the optimization problem is NP-hard in general, but is polynomially solvable if the reward posterior is Gaussian and there is no uncertainty in the transitions. Still, second order approximations yield a tractable approximation in the general case, if there is a Gaussian prior to the reward and a Dirichlet prior to the transitions.

The above works address policy optimization and evaluation given an exogenous state sampling procedure. It is of interest to consider the exploration-exploitation problem in reinforcement learning (RL) from the sample complexity perspective as well. While the Bayesian approach to model-based RL offers an elegant solution to this problem, by considering a distribution over possible models and acting to maximize expected reward, the Bayesian solution is intractable for all but the simplest problems; see, however, stochastic tree search approximations in **?**. Two recent papers address the issue of complexity in model-based BRL. In the first paper, **?** present a simple algorithm, and prove that with high probability it is able to perform approximately close to the true (intractable) optimal Bayesian policy after a polynomial (in quantities describing the system) number of time steps. The algorithm and analysis are reminiscent to PAC-MDP (e.g., **??**) but it explores in a greedier style than PAC-MDP algorithms. In the second paper, **?** present an approach that drives exploration by sampling multiple models from the posterior and selecting actions optimistically. The decision when to re-sample the set and how to combine the models is based on optimistic heuristics. The resulting algorithm achieves near optimal reward with high probability with a sample complexity that is low relative to the speed at which the posterior distribution converges during learning. Finally, **?** derive a PAC-Bayesian style bound that allows balancing between the distribution-free PAC and the data-efficient Bayesian paradigms.

5 Summary and Discussion

While Bayesian Reinforcement Learning was perhaps the first kind of reinforcement learning considered in the 1960s by the Operations Research community, a recent surge of interest by the Machine Learning community has lead to many advances described in this chapter. Much of this interest comes from the benefits of maintaining explicit distributions over the quantities of interest. In particular, the exploration/exploitation tradeoff can be naturally optimized once a distribution is used to quantify the uncertainty about various parts of the model, value function or gradient. Notions of risk can also be taken into account while optimizing a policy.

In this chapter we provided an overview of the state of the art regarding the use of Bayesian techniques in reinforcement learning for a single agent in fully observable domains. We note that Bayesian techniques have also been used in partially observable domains (**?????**) and multi-agent systems (**???**).